Resumen

El objetivo del proyecto es desarrollar una aplicación para dispositivos que funcionen con el sistema operativo Android que simule un osciloscopio muy básico. Así se puede conseguir un osciloscopio accesible a todo el mundo y que, a diferencia de los programas ejecutables por medio del PC, no requiera de instalación de librerías para que funcione.

Como hardware se ha utilizado una placa Bus Pirate, con la versión 3.6, para conseguir tal objetivo. Los pasos a seguir en el proyecto son el estudio de las especificaciones de la placa, desarrollar un código que garantice una correcta lectura de datos del puerto serie de la placa y lo transmita mediante USB al dispositivo y, por último, integrar este programa en un osciloscopio y adaptarlo para que funcione tal y como se quiera.

Del estudio de la placa se ha podido conocer que ésta tiene dos modos de acceso: el modo de acceso por terminal de usuario y el modo de acceso binario. Como en el primero los datos que se envían son en formato texto hace que la velocidad de lectura de datos resulte muy lenta para el desarrollo de un osciloscopio. Por tanto, se decide trabajar en el modo de acceso binario en donde esta información es en formato de bytes.

En el modo acceso binario interesa trabajar en el modo Bit Bang, ya que proporciona un control directo sobre los pines de la placa con tan solo mandar un byte, proceso que resulta muy rápido. Para entrar en este modo, se envía veinte veces el byte 0x00 y una vez dentro se puede hacer mediciones a través de la sonda ADC (conversor analógico digital del microcontrolador interno de la placa Bus Pirate) enviando a la placa el byte 0x14 (una vez) o el byte 0x15 (varias veces). Lo que la placa nos devuelve es una pareja de bytes en formato Motorola, es decir, los 8 bits más significativos van primero. Esto se tiene en cuenta a la hora de desarrollar el código para que no se mezcle una lectura con otra.

Una vez realizado el estudio de la placa se procede al desarrollo del programa. En un primer instante, se había pensado en escribir este código en Python y desarrollar la aplicación en una librería de Python llamada Kivy. Al crear un primer intento en el que se intentaba acceder al USB del terminal móvil/Tableta se observó que no se tenían los permisos suficientes y que, por tanto, para poder acceder a él se tenía que “rootear” el dispositivo, hecho que contradice uno de los objetivos del proyecto que era la accesibilidad del osciloscopio sin necesidad de ninguna instalación adicional.

Por tanto, se decide desarrollar el programa en el lenguaje de programación Basic for Android (B4A a partir de ahora), que es un lenguaje basado en Visual Basic y que no requiere de permisos adicionales para acceder al USB del terminal ni ningún otro tipo de dependencia.

Una vez se ha desarrollado el código para que se recojan los datos de la sonda ADC del Bus Pirate y se han convertido a Voltios para que resulte funcional en el osciloscopio, se integra en el código del osciloscopio que se ha tomado como base. Se eliminan las curvas adicionales, ya que sólo se necesita una variable que mostrar (voltaje) y se trabaja sobre el modo Single Shot, en el que se toman un número determinado de muestras y una vez tomadas todas esas muestras se dibuja en la pantalla, siendo el eje Y el voltaje y el eje X el tiempo.

Sumario

ETSEIB

1. Prefacio

1.1. Contextualización y motivación del proyecto

Un osciloscopio común es un aparato que no es accesible económicamente a todo el mundo y menos si se necesita en grandes cantidades, como podría ser en el ámbito académico (por ejemplo, en un instituto en el que no se dispusiese de mucho presupuesto sería inviable hacer una inversión de tales cantidades para que sus alumnos puedan trastear con el osciloscopio e introducirse en el mundo de la ingeniería).

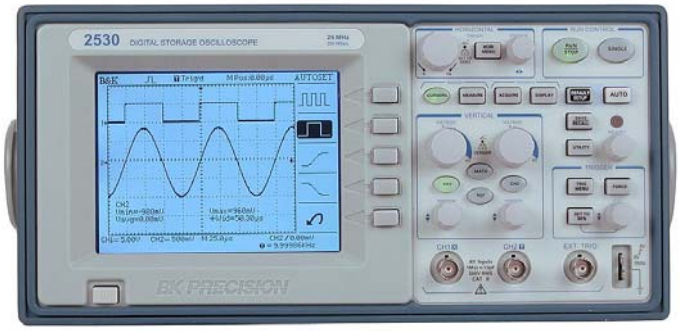

Figura 1. 1 – Osciloscopio físico. Fuente: directindustry

Una alternativa a los tipos de osciloscopios comunes son programas que se pueden ejecutar desde un PC y que simulen las funciones de un osciloscopio, así se podría disponer de uno desde cualquier lugar en el haya un ordenador. El problema viene en que este tipo de programas requieren de instalaciones de librerías adicionales para que funcionen. En el caso de un PC personal esto no supondría un hándicap insalvable pero en el caso de cualquier PC en el que se requiera permisos de administrador para instalar librerías (como puede ser el caso de los PC habilitados en una biblioteca o en un aula de informática de la universidad…) no se podría ejecutar el programa y, por tanto, de disponer del osciloscopio.

El proyecto nace entonces para solventar los problemas antes expuestos. Se ha de desarrollar un osciloscopio que sea totalmente independiente (excepto el hardware que hace posible tomar las mediciones de voltaje que eso siempre se necesitará), económico, portátil y accesible a todo el mundo.

Así pues, si se desarrolla una aplicación para dispositivos móviles/Tablet, se cumplen todos los requisitos antes descritos (ya que una aplicación no requiere de ninguna instalación adicional aparte de la aplicación en sí para ser ejecutada, es económica, portátil y accesible

a todo el mundo que disponga, por ejemplo, de un terminal móvil). Para generalizar más su posible uso se ha decidido desarrollar esta aplicación para el sistema operativo Android, que es el más extendido actualmente.

1.2. Requerimientos previos

Para la total comprensión del proyecto son necesarios unos ligeros conocimientos de electrónica así como de programación en Python y en Visual Basic, lenguaje en el que se basa B4A.

2. Introducción

2.1. Objetivos del proyecto

El objetivo principal del proyecto es poder desarrollar una versión de osciloscopio que permita ser usado fácilmente en cualquier lugar, sin necesidad de invertir mucho dinero en su compra y sin tener que hacer instalaciones adicionales a la del propio osciloscopio (quedando excluidas así las versiones de osciloscopio que pueden ser usadas en un PC pero que necesitan muchas veces de instalaciones de librerías adicionales que no siempre son permitidas por el administrador del equipo).

Así pues, con la finalidad de cumplir todos estos objetivos, se ha decidido desarrollar una aplicación en Android de un osciloscopio que funciona conectando una placa de hardware comercial denominada Bus Pirate al puerto USB del terminal en el cual se ejecuta la aplicación.

Por lo tanto, el objetivo final del proyecto es desarrollar una aplicación en Android que, a través de una placa Bus Pirate, se comporte como un osciloscopio básico que pueda medir, por ejemplo, una señal escalada de voltaje de red.

Como objetivos parciales se podrían enumerar tres fácilmente, que serían los pasos a seguir para alcanzar la meta planteada:

- Hacer un estudio exhaustivo de las especificaciones de la placa Bus Pirate para poder sacar el máximo provecho de sus características para usarlo en el osciloscopio.
- Desarrollar un programa que permita leer los datos que nos manda la placa mediante el USB y convertir esta información en datos útiles para utilizarlos en el osciloscopio.
- Coger un programa de osciloscopio como base y modificarlo integrando el programa que se ha desarrollado previamente que nos permite leer los datos de la placa para así tener un osciloscopio en que los datos de la curva se base en lo que nos envíe el Bus Pirate.

2.2. Alcance del proyecto

El proyecto busca básicamente centrarse en la problemática de la comunicación entre el USB de un terminal móvil/Tablet y el puerto serie de una placa como Bus Pirate y sacar datos de ella para poder ser utilizados en diversas aplicaciones como en un osciloscopio. Una vez desarrollado el programa que permita esta comunicación y que convierta los datos adecuadamente, el código se puede adaptar a diversas funcionalidades.

Obviamente, para poder desarrollar este código, se ha tenido que hacer un estudio de las especificaciones de la placa y saber cómo funciona, por lo tanto, esto va implícito en el objetivo de desarrollar dicho código.

Una vez realizada esta tarea se pretende integrarla y hacer funcionar un osciloscopio muy básico que permita la lectura de voltaje y lo muestre en un gráfico. A partir de aquí, se puede ir desarrollando todavía más la aplicación del osciloscopio pudiéndole añadir más funcionalidades que las básicas.

3. Bus Pirate

3.1. Introducción

Enfrentarse a un microchip nuevo puede ser una ardua tarea. “Breadboarding” un circuito, escribir el código, buscar el programador, etc…

Bus Pirate [Fig. 4.1] es una multi-herramienta de bajo coste, de código abierto, que permite comunicarse con otros aparatos electrónicos de forma fácil. Bus Pirate es una interfaz de bus universal que se comunica con la mayoría de los chips desde un terminal serie de PC, eliminando así la necesidad del trabajo previo requerido cuando se trabaja con chips nuevos o desconocidos.

Bus Pirate nace como un proyecto iniciado por Ian Lesnet, que había trabajado previamente con “Seed Studio” para hacer disponible hardware abierto. En 2009 lanza el proyecto *DangerousPrototypes.com,* donde vende esos proyectos de código abierto, entre los que se encuentra el Bus Pirate.

Bus Pirate puede realizar una gran variedad de funciones de test de equipos tales como medir voltajes y frecuencias así como generar señales PWM.

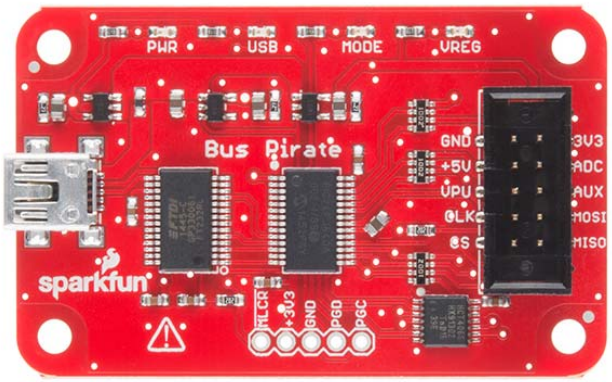

Figura 3. 1 – Bus Pirate v3.6. Fuente: SeedStudio.

3.2. Elección versión hardware Bus Pirate

Actualmente existen 4 versiones del hardware, siendo la versión 4 la más reciente de ellas. En la tabla adjunta se muestra una comparación entre las versiones v3 y v4 del Bus Pirate:

Característica	***Bus Pirate v4***	***Bus Pirate v3***
Confiabilidad	Experimental, nuevo	Bien probado, confiable
Soporte y desarrollo	Activo	Activo
Memoria Flash	256 K	64 K
Memoria RAM	16 K	4 K
Conexión USB	PIC integrado USB	FTDI USB a chip serial
Pines IO	7 IO + accesorios	5 IO + accesorios
Almacenaje configuración, “built-in” demo EEPROM	4K	n/a
Suministro resistencias Pull-up	3.3V, 5V, externo	Sólo externo
Botones	2	0
AVR programador (STK500v2)	Integrado	Firmware externo
XSVF player (JTAG)	Integrado	Firmware externo

Tabla 3. 1 – Comparación de versiones de Bus Pirate. Fuente: DangerousPrototypes.

Claramente se puede observar que en temas exclusivamente de hardware, la versión 4 de Bus Pirate es claramente superior a la versión 3 del mismo. Empezando por un incremento de la memoria Flash de 64 K a 256 K (por lo que nos permite almacenar más datos en la placa) y por el incremento de 4K a 16K de la memoria RAM (lo que se traduce en más programas/aplicaciones más complejas funcionando en ella).

La conexión USB también cambia respecto a las versiones de la placa, siendo la de la v4 una conexión USB más rápida (una velocidad máxima de 12MBPS, pero con un límite realista de 1MBPS) que la utilizada en la v3 (vel. Máxima: 4MBPS, límite realista: 115 KBPS).

Bus Pirate v4 tiene una pequeña memoria I2C EEPROM que almacena configuraciones, modos de arrancar, etc… que la v3 no tiene. Además, la v4, dispone de dos botones. Uno es un botón de reset, para resetar la placa y el otro es un botón para borrar modos guardados en la configuración de la EEPROM.

Bus Pirate v4 dispone también de resistencias pull-up que pueden ser cambiadas entre 3.3 V, 5 V y una fuente externa mediante la interfaz de software, v3 sólo soporta una fuente externa de voltaje.

Para terminar, la v4 dispone de dos pines extra (7 frente a los 5 pines disponibles en la v3) que resultan ser muy útiles para el modo OpenOCD JTAG. Por desgracia, tanto el modo OpenOCD JTAG como las funciones integradas: programador AVR (STK500v2) y programdor XSVF player (JTAG) actualmente no funcionan en el v4, por lo que pierden estas características a la hora de elegir una de las versiones.

Aún con todas las mejoras de hardware que obviamente dispone la v4 respecto la v3 es innegable que en cantidad de recursos y de apoyo de los usuarios es mucho mayor en la v3 que en la v4 por el simple hecho de ser esta última mucho más reciente. Además, v4 es todavía considerada como una versión tester por lo que hay muchos errores que corregir.

Las mejoras de hardware se podrían utilizar para hacer un osciloscopio más complejo (por la mayor memoria Flash y memoria RAM), con una velocidad de lectura todavía más rápida (tipo USB), pero realmente no es algo estrictamente necesario, ya que habría que sacrificar la cantidad de apoyo en forma de programas que hay para la v3 y su mayor estabilidad. Sí que podría haber sido de mucha utilidad las mejoras referentes al botón de reset (en la v3 cada vez que se quiere hacer un reset se debe desconectar la placa del USB y volverla a conectar, con la pérdida de tiempo que eso conlleva), así como la introducción de una memoria para la configuración haciendo así que la placa inicialice en el modo que nos interesase automáticamente una vez configurado. Son detalles menores que habrían hecho

el proyecto más fácil pero que se puede llegar a una solución, algo primitiva, mediante software pero que sigue siendo válida.

Por todo lo anteriormente expuesto, se ha decidido escoger la versión v3 del hardware de Bus Pirate para realizar el proyecto.

Nótese que dentro de la versión de hardware v3 hay diversas subversiones (concretamente v3a, v3b, v3.5 y v3.6, siendo esta la última versión de la v3). Debido a que las mejoras que hay entre subversiones son puramente estéticas y que la v3.6 engloba todas las subversiones anteriores en cuanto a especificaciones y funcionalidad, se ha decidido escoger la v3.6 como la placa con la que se trabajará en el proyecto.

3.3. Hardware

Muchas de las funcionalidades del Bus Pirate giran en torno a la comunicación mediante protocolos serie. El Bus Pirate se puede comunicar mediante los protocolos 1-cable, 2-cables, 3- cables, UART, I2C, SPI y HD44780 LCD. También dispone de un modo bitbang en donde personalizar opciones.

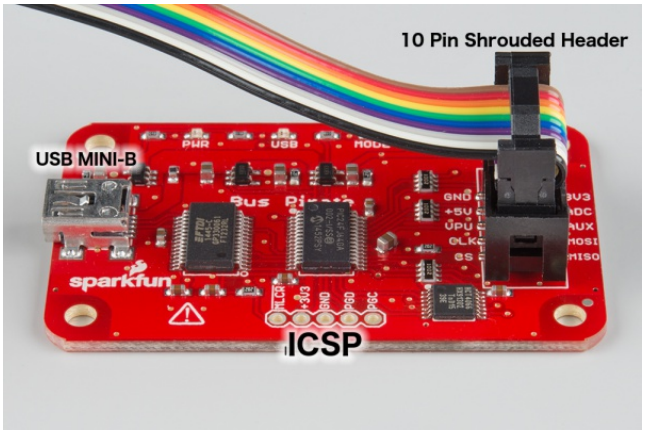

Figura 3. 2 – Puertos de conexión disponibles en Bus Pirate. Fuente: Seedstudios.

Tal y como se puede apreciar en la Fig. 4.2, Bus Pirate dispone de 3 puertos. El primero de ellos, el puerto ICSP es utilizado para programar el microcontrolador directamente. Debido a que se dispone de un bootloader y a la utilidad de poder reflashear el microcontrolador, no se debería usar nunca este puerto.

El segundo puerto es un USB tipo mini-B. Si conectamos este puerto al PC mediante un cable USB A – USB mini-B, este proporciona la energía necesaria a la placa y permite comunicarnos con ella de forma no intrusiva a diferencia del puerto ICSP.

El tercer y último puerto es el que se utilizará para conectar el Bus Pirate al sistema que se esté desarrollando o con el que se esté trabajando, ya sea para leer un voltaje, una frecuencia, etc…Es un puerto que dispone de un conector tipo *0.1" pitch 2.5 pin header*. Para comunicar este puerto y el sistema objetivo se utilizará un cable como el de la Fig. 4.3:

Figura 3. 3 – Cable conector 2x5 pines. Fuente: BricoGeek.

A continuación se adjuntan una serie de figuras, en donde se describe y se relacionan las funciones de los pines del bus Pirate con los colores del cable que hace de conector entre la placa y el sistema que estemos desarrollando. Las fuentes de alimentación de cada uno de los pines de la placa puede ser activada o desactivada mediante software y cada uno de ellos puedes suministrar hasta 150 mA al sistema:

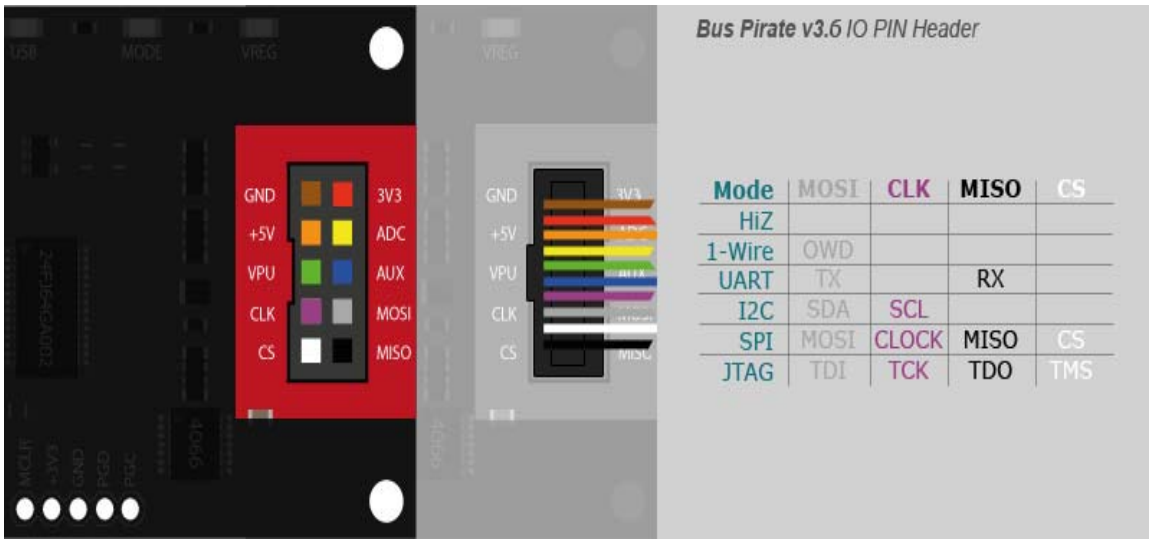

Figura 3. 4 – Relación pines-función. Fuente: DangerousPrototypes.

Nombre PIN	**Color del cable**	**Descripción (Bus Pirate es el master)**
GND	Marrón	Tierra, conectado a la tierra
+3.3V	Rojo	+3.3 V fuente alimentación commutable
+5.0V	Naranja	+5 volt fuente alimentación commutable
ADC	Amarillo	Sonda medición voltaje (max 6V)
VPU	Verde	Entrada de tensión para resistencias pull-up integradas (0V to 5V).
AUX	Azul	E/S auxiliar, sonda de frecuentcia, modulador de ancho de pulso

CLK	Lila	Señal de reloj (I2C, SPI, JTAG, KB)
MOSI	Gris	Master data out, slave in (SPI, JTAG), Serial data (1-Wire, I^2C, KB), TX (UART)
CS	Blanco	Selector de chip (SPI), TMS (JTAG)
MISO	Negro	Master data in, slave out (SPI, JTAG) RX (UART)

Tabla 3. 2 – Relación color cable – función. Fuente: DangerousPrototypes.

3.3.1. Conexión USB

El Bus Pirate se comunica con el PC a través del puerto USB FTDI a UART incorporado en la placa.

El FT23RL es uno de los circuitos integrados más comúnmente usados para convertir señales USB a señales UART. Este proceso es muy útil, ya que te permite comunicar y actualizar código de un microcontrolador sin necesidad de un programador externo. La placa utilizada en el Bus Pirate v3.6 es la llamada SparkFun y lleva incorporada una placa llamada FTDI basic, que conecta los pines de forma conveniente en el FT232RL para realizar las acciones antes mencionadas.

Figura 3. 5 – Placa SparkFun. Fuente: Sparkfun.

3.4. Software

Para permitir la comunicación entre el USB de la placa y el PC hemos de instalar en este último los drivers necesarios para que el PC los reconozca. En Windows (que es el sistema operativo con el que se está trabajando) estos drivers no están incluidos y, por tanto, hemos de obtenerlos a través de la página web de FTDI (se llama VCP “Virtual COM Port”).

Bus Pirate tiene dos modos de acceso: el modo de acceso por terminal de usuario y el modo de acceso binario.

3.4.1. Modo terminal de usuario

En este modo se accede al Bus Pirate a través de líneas de comando mediante un terminal.

Bus Pirate siempre se inicializa en el modo de alta impedancia (Hi-Z), un modo seguro con todas las salidas desactivadas, pensado para proteger cualquier dispositivo conectado al microcontrolador de funcionar en condiciones fuera de su rango de especificaciones, evitando así que se dañen por un mal uso. Desde ahí se puede seleccionar el modo a utilizar por Bus Pirate con un protocolo específico mediante un menú interactivo.

Este modo es una muy buena opción para usuarios que empiezan usando el Bus Pirate, ya que es más intuitivo y más gráfico. El problema que conlleva utilizar este modo es que al usar texto como datos, el envío y la lectura de estos datos se hace, obviamente, de forma muy lenta, o al menos, mucho más lento que haciéndolo mediante un valor binario en bruto, por lo que tardaríamos demasiado en recibir los voltajes y no sería viable usarlo para hacer un osciloscopio portátil. Sin embargo, este modo es útil, en este caso, para comprobar la funcionalidad de la placa, así como asegurar que esta está actualizada.

Todo lo necesario para usar este modo es un terminal de comunicaciones serie. En este caso se usará un ejecutable del terminal Putty. Para poder obtener información de la placa hemos de configurar Putty para una conexión tipo serial, indicando cuál es el puerto USB en el que el Bus Pirate está conectado (en este caso, el COM5) y la velocidad a la cual se conectará (115200 BPS), tal y como se muestra en la siguiente imagen:

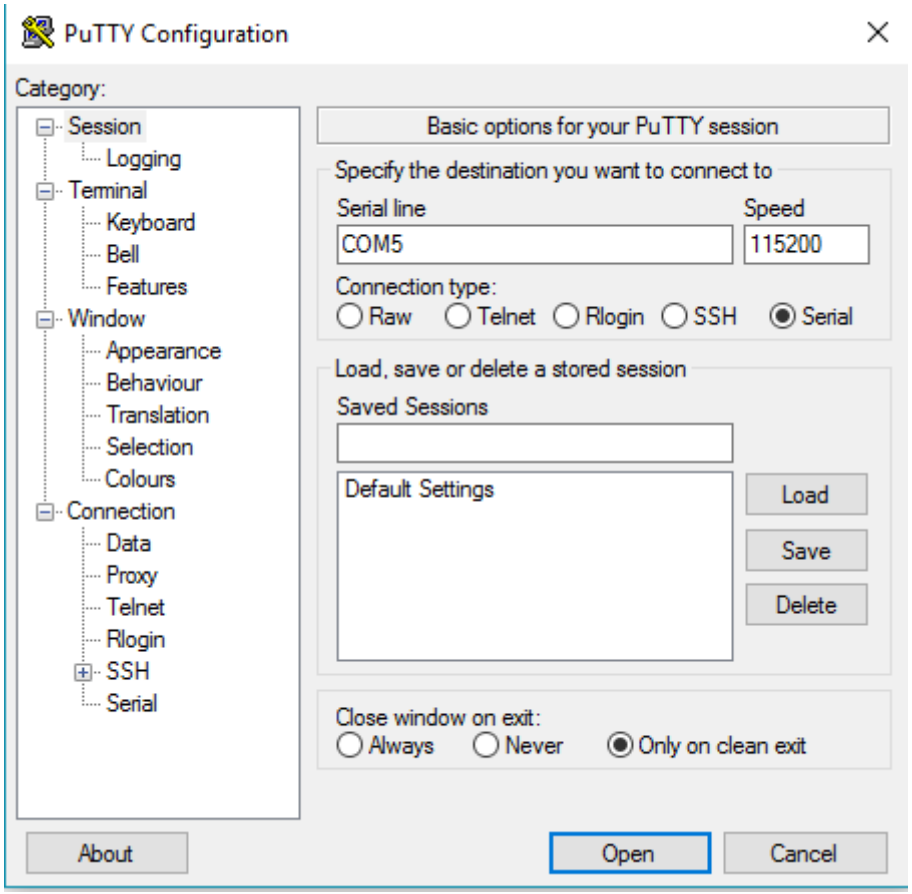

Figura 3. 6 - Pantalla configuración Putty. Fuente: Propia.

Una vez iniciado el terminal, si escribimos "?" nos aparecerá la lista de comandos que podemos utilizar. Como vemos al final de la pantalla del terminal, el Bus Pirate ha entrado automáticamente en el modo alta impedancia (HiZ), tal y como se ha mencionado anteriormente:

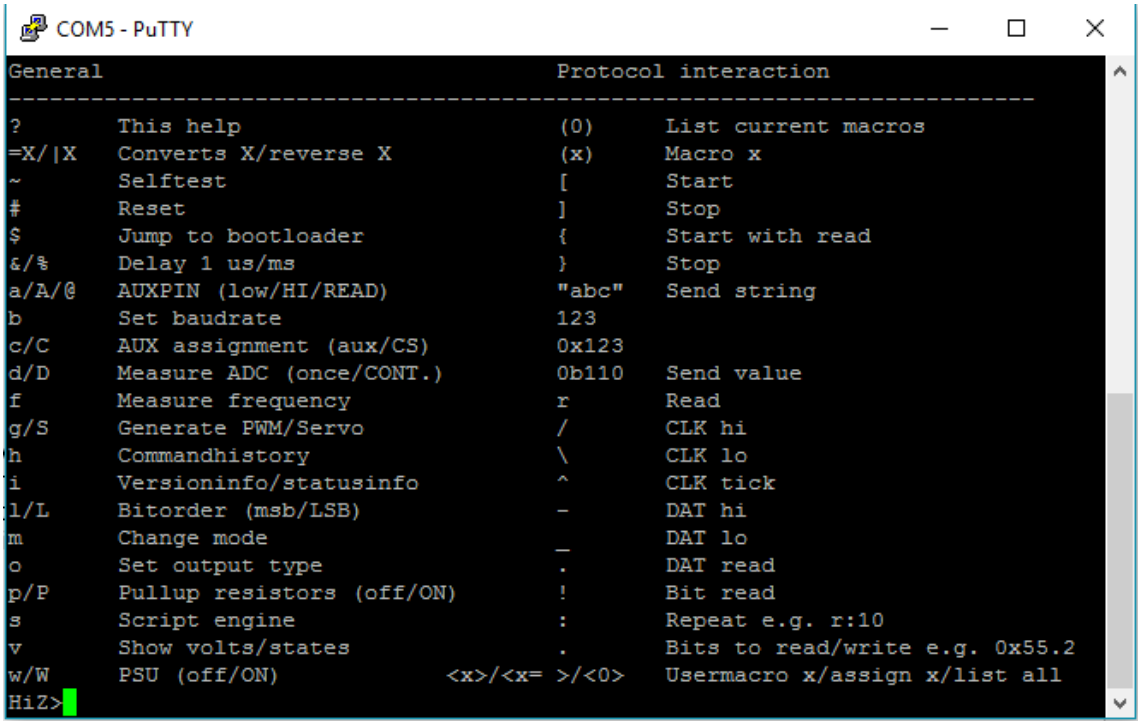

Figura 3. 7 - Captura lista comandos BP. Fuente: Propia.

Lo primero que convendría comprobar es que se tiene actualizada la placa a su última versión de firmware, que a fecha del proyecto es la 5.10, para evitar posibles bugs que hubiese en anteriores versiones y usar la placa con todas las funcionalidades disponibles mediante actualizaciones de software. Para comprobar el firmware del Bus Pirate en el terminal introducimos el comando “i”, tal y como nos indica el menú de ayuda:

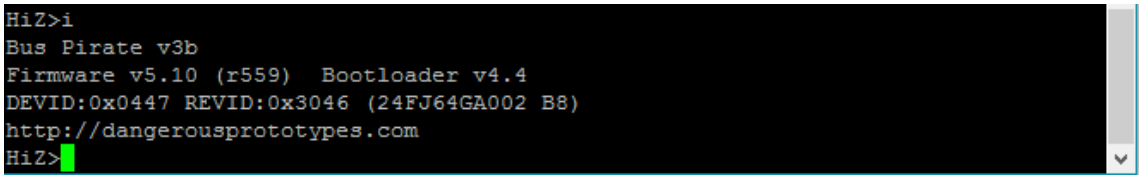

Figura 3. 8 - Versión firmware. Fuente: Propia.

Se puede observar que la placa está convenientemente actualizada. En caso de que no lo hubiese estado, actualizar el firmware es tan sencillo como ejecutar el paquete de instalación mediante el sistema de comandos.

Se procede ahora a comprobar que el Bus Pirate realiza su función sin problemas. En el caso de este proyecto, lo que necesitamos es que el pin que permite medir el voltaje (ADC) que corresponde al cable amarillo funcione correctamente. Para ello vamos a conectar el cable a una fuente de voltaje externa que nos permita medirlo. Es importante recordar que el voltaje máximo que se puede medir es 6V, por lo que conviene no sobrepasarlos. Para hacer la prueba se ha conectado la placa a una fuente de alimentación variable controlable, conectando el positivo en el cable amarillo (ADC) y el negativo al cable marrón (Tierra). Observando el menú de ayuda del terminal vemos que para medir el voltaje tenemos dos opciones representadas por una d y una D. Ambas miden el voltaje conectado al cable amarillo pero si se introduce una d, la medición se hará una sola vez. Sin embargo, si se introduce una D, la medición del voltaje será continua, es decir, se seguirá midiendo el voltaje de forma indefinida (por lo que se irán pasando valores) hasta que se cierre el programa. Lógicamente si para lo que se quiere esa medición de voltaje es para usarla en un osciloscopio, el modo más obvio de usar es el modo continuo.

Se conecta los cables a la fuente de alimentación y se le aplican 3,3 V aproximadamente a la placa, introduciendo el comando “d” para que nos haga una sola medición de voltaje, se obtiene:

Figura 3. 9 - Medición única de voltaje. Fuente: Propia.

Se puede comprobar que efectivamente mide aquello que la fuente nos está dando y que lo hace una sola vez, tal y como se había previsto. A continuación, se hace lo mismo pero variando la fuente con tal de probar la función de medición continua. En este caso, se varía la fuente desde 3,8V hasta 0V:

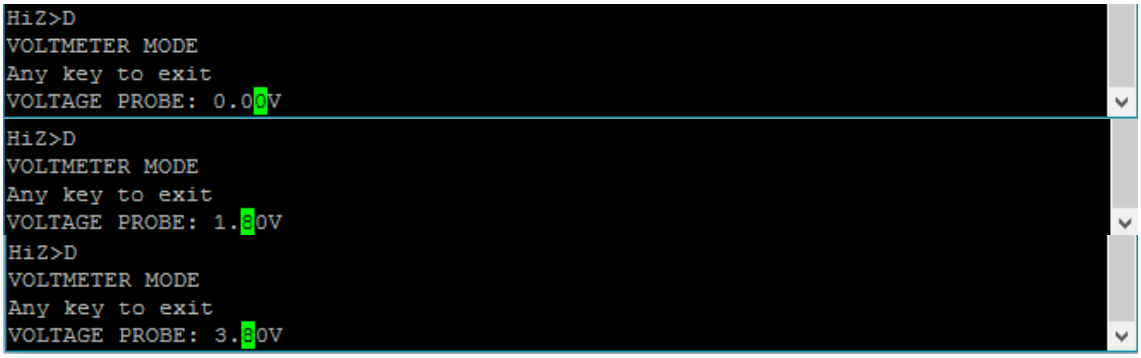

Figura 3. 10 - Mediciones voltaje continua de 3.8 a 0 V. Fuente: Propia.

Como el voltaje recibido se actualiza sobre la misma línea en el terminal se han hecho tres capturas para demostrar cómo ha habido esa variación desde los 3,8 V hasta los 0V.

Por tanto, tras estas pruebas iniciales en el modo terminal de usuario, se ha determinado que estamos usando la última versión disponible del software Bus Pirate así como que funciona el pin ADC para medir tensiones tanto la función en que recoge el dato una sola vez como la versión que recoge múltiples tensiones hasta que se cierra el programa. Pero, tal y como se ha mencionado anteriormente, se ha de utilizar un modo que no resulte tan lento, ya que la velocidad es determinante a la hora de diseñar un osciloscopio. Para ello, se utilizará el modo de acceso binario.

3.4.2. Modo de acceso binario

El modo de acceso binario puede ser usado con software o scripts. Este modo tiene diferentes protocolos pero el que interesa para el proyecto es el llamado “Bitbang mode”. El modo Bitbang proporciona un control directo sobre los pines del Bus Pirate y su hardware usando un protocolo de un solo byte para todos los comandos. El estado inicial de la placa, al igual que en el modo terminal de usuario, es el modo HiZ, es decir, el modo de alta impedancia en que todas las salidas están desactivadas.

Características:

- Para entrar en el modo binario Bitbang se ha de enviar **20 veces** el byte 0x00 al terminal de usuario. Se puede saber si se ha entrado en el modo si el terminal te devuelve la cadena de caracteres “**BBIO1”**.
- Para resetear el Bus Pirate y, por tanto, salir del modo binario Bitbang (ya que en el versión v3.6 no se dispone de un botón físico de reset) se debe enviar el byte 0x0F

- Otros protocolos (tales como SPI, I2C, UART…) pueden ser accedidos a través del modo Bitbang. Para volver al modo Bitbang solo es necesario enviar 0x00.
- Hay un pequeño retraso de unos **5µs** entre las actualizaciones de un pin.

Comandos:

En el modo binario Bitbang se pueden utilizar los siguientes comandos según el byte que se envíe al terminal (entre paréntesis se escribe el mismo número en hexadecimal en vez de binario ya que será de la forma en la que se utilizará más adelante):

1) 00000000 (0x00) Reset, responde "BBIO1": resetea el Bus Pirate a modo Bitbang. Si se envía 20 veces se entra en el modo binario Bitbang desde el modo terminal de usuario. Si se ha accedido a otro modo de protocolo (como SPI, UART…) te devuelve al modo Bitbang.
2) 00000001 (0x01) Entra en modo binario SPI, responde "SPI1"
3) 00000010 (0x02) Entra en modo binario I2C, responde "I2C1"
4) 00000011 (0x03) Entra en modo binario UART, responde "ART1"
5) 00000100 (0x04) Entra en modo binario 1-Wire, responde "1W01"
6) 00000101 (0x05) Entra en modo binario raw-wire, responde "RAW1"
7) 00000110 (0x06) Entra en modo OpenOCD JTAG
8) 00001111 (0x0F) Reset Bus Pirate. Realiza un completo reset de hardware del Bus Pirate (por lo que el Bus Pirate sale del modo binario Bitbang y vuelve al modo terminal de usuario) y éste responde con 0x01. La versión de hardware y firmware de la placa son mostradas (equivalente al comando "i" en el modo terminal de usuario).
9) 0001000x (0x1x) Bus Pirate auto-tests. Tests incluidos dentro del Bus Pirate para comprobar el correcto funcionamiento de los pines y periféricos de la placa.
10) 00010010 (0x12) Configura y activa el PWM output en el pin AUX. Requiere una secuencia de configuración de 5 bytes. Responde 0x01 después de que la secuencia se reciba. El PWM sigue activo incluso después de salir del modo Bitbang.
11) 00010011 (0x13) Borra/ Desactiva PWM output. Responde 0x01.
12) 00010100 (0x14) Recoge la medida de voltaje de sonda del Bus Pirate. Devuelve una lectura ADC de 2 bytes, los 8 bits más significativos siempre van primero. Para obtener la medida de voltaje se puede utilizar la siguiente ecuación:

$$V = (ADC/1024)*6{,}6$$

13) 00010101 (0x15) Recoge datos ADC (2 bytes, los 8 bits más significativos van primero) tan rápido como la UART lo permite. Una nueva lectura no se realiza hasta que

la previa no se ha terminado de transmitir al PC, esto previene una distorsión de tiempo del buffer.

14) 00010110 (0x16) Mide la frecuencia del pin AUX. Devuelve 4 bytes de frecuencia, el byte más significativo va primero.

15) 010xxxxx (0xxx) Configura los pins como inputs/entradas (1) o outputs/salidas (0) estos pins están mapeados del siguiente orden: AUX/MOSI/CLK/MISO/CS. El Bus Pirate responde con un byte mostrando el estado actual de los pins. Así, por ejemplo, si enviamos 01010010 los pins AUX y MISO están configurados como inputs y los pins MOSI, CLK y CS lo están como outputs.

16) 1xxxxxxx (0xxx) Enciende (1) o apaga (0). Los últimos 7 bits del comando controlan el encendido y el apagado de los pines del Bus Pirate. Éstos están mapeados en el siguiente orden: 1/POWER/PULLUP/AUX/MOSI/CLK/MISO/CS. El Bus Pirate responde cada actualización con un byte en el mismo formato que muestra el estado actual de los pines.

4. Python

El núcleo para un buen funcionamiento del osciloscopio portátil es obtener una buena comunicación entre el USB y el puerto serie de la placa. Para ello, se escribirá primero el código en Python y luego se traducirá a un lenguaje de programación apto para ser usado en Android, que es el objetivo del proyecto.

El motivo de usar Python como lenguaje principal es la sencillez de su uso, así como el conocimiento que se tiene del mismo y que la idea principal que se tiene es usar Kivy, que es una librería de Python de código abierto para desarrollar la aplicación Android, por lo que si ya se tiene el código escrito en Python será muy fácil trasladar esto a Kivy (básicamente se tratará de adaptar la parte gráfica del osciloscopio).

4.1.1. Código USB Serial Python

Al escribir el código lo primero que se hace es importar las librerías. Así pues, para realizar la conexión con la UART de la placa, necesitaremos la librería serial, que nos permitirá enviar bytes a la placa mediante comunicación serie asíncrona. También importamos la librería time, librería que nos permitirá añadir algún que otro retardo puro (delay) para no saturar la placa.

```
import serial
import time
```

Una vez importadas las librerías necesarias, se pasa a configurar la conexión serial:

```
ser = serial.Serial(
    port='COM5',
    baudrate=115200,
    parity=serial.PARITY_NONE,
    stopbits=serial.STOPBITS_ONE,
    bytesize=serial.EIGHTBITS
)
```

Lo primero que se indica es el puerto en donde está conectada la placa, que en este caso concreto es el COM5 (en este programa, la placa va conectada directamente a uno de los puertos USB del PC) así como la velocidad, número de unidades de señal por segundo, de envío (baudrate) a 115200, que es la máxima velocidad a la que puede trabajar el Bus Pirate.

Una vez configurado el puerto serie, se necesita que el Bus Pirate entre en modo Bit Bang. Tal y como se ha explicado anteriormente (véase el apartado 4.4.2), para ello se necesita enviar el byte 00000000 (0x00) veinte veces. Para tal fin, se crea un bucle en que mediante la función ser.write se envía este byte veinte veces:

```
for i in range(1,21):
    ser.write([0x00])
```

Al haber enviado el byte 0x00 veinte veces, el Bus Pirate entra en el modo Bit Bang.Se puede comprobar que está en este modo de dos formas distintas: primero, el LED MODE de la placa se enciende una vez está en el modo Bit Bang, por lo que si una vez se envían los veinte bytes 0x00 se aprecie que este LED se enciende se habrá entrado en el modo correctamente. Segundo, a través del terminal se puede observar que la placa responde con “BBIO1”. Si es así, no hay duda de que se ha entrado correctamente.

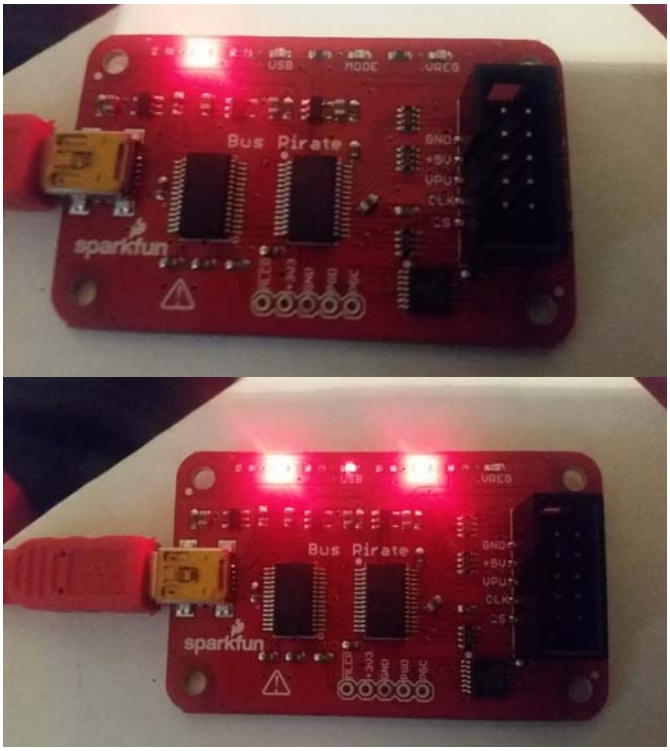

Figura 4. 1 - Placa en antes y después de entrar modo Bit Bang. Fuente: Propia.

Al haber entrado en el modo Bit Bang ya se puede acceder a las mediciones de datos ADC mediante el envío del byte 00010101 (0x15) a la placa (se recuerda que recoge de manera continua datos ADC y devuelve 2 bytes, los 8 bits más significativos van primero). Antes de ello, sin embargo, se introducirá un pequeño retardo (delay) de 0,1 s para que haya espacio de tiempo entre entrar en el modo Bit Bang y empezar a hacer las mediciones:

```
time.sleep(0.1)
ser.write([0x15])
```

Para asegurar de que realmente se están haciendo las mediciones, se escribirán unas últimas lineas de código para imprimir por el terminal las mediciones. Como estas serán datos en binario se aplicará una conversión a hexadecimal para que sean más entendibles:

```
while 1 :
    if ser.inWaiting() > 0:
        print(hex(ord(ser.read(1))))
```

Se verifica el código y se procede a hacer unas mediciones dentro de la misma placa para comprobar que realmente la sonda ADC está midiendo voltajes.

5. Kivy

Kivy es una librería de código abierto de Python que permite desarrollar aplicaciones de manera más o menos sencilla y bastante cómoda si se está familiarizado con el lenguaje Python. Básicamente, se tratan de librerías de Python que añaden mayoritariamente los elementos gráficos de una aplicación. En kivy se suele separar la parte lógica de la parte gráfica. Así, por ejemplo, si se está creando un juego tipo Pong, en el archivo .py principal se definirán propiedades como la velocidad y la física de la bola, el movimiento de las columnas etc… mientras en el archivo de mismo nombre y con extensión .kv se definirá todo el tema gráfico como la dimensión de la bola, el color, longitudo de las columnas, etc.

Una aplicación hecha con kivy tiene la siguiente estructura básica:

```
import kivy
kivy.require('1.0.6') # replace with your current kivy version !

from kivy.app import App
from kivy.uix.label import Label


class MyApp(App):

    def build(self):
        return Label(text='Hello world')


if __name__ == '__main__':
    MyApp().run()
```

Figura 5. 1 - Estructura básica kivy. Fuente: Kivy.org.

Obviamente, lo primero que se ha de importar es la librería kivy, y una vez importada esta, se importa la clase App que será la clase de la aplicación. Después, debe ser importado el módulo uix que contiene el layout básico de la aplicación así como la interfaz de usuario.

Tras importar lo básico para que tener una aplicación, la estructura a seguir es crear una clase App que será la aplicación en sí y dentro el "def build (self)" que es lo que se ejecutará una vez se inicie la aplicación. En el ejemplo expuesto en la Fig. 6.1 lo que se ejecuta es que aparezca una etiqueta con el texto: "Hello world".

Por último, la última línea de código es la que permite ejecutar la aplicación.

5.1.1. Código USB Serial Kivy

Hacer una conexión satisfactoria entre móvil y Bus Pirate mediante una aplicación es el primer objetivo que se ha de cumplir para continuar trabajando en el osciloscopio. Para ello se dispone de un código sencillo que establece esta conexión y que dibuja círculos a partir de los datos que recoge del puerto serie cada 0,5s (esto se consigue a través del módulo Clock)

```
# Basic class Float Layout
class Test1(FloatLayout):

    def __init__(self, **kwargs):
        super(Test1, self).__init__(**kwargs)

        # Set the timer for redrawing the screen
        refresh_time = 0.5
        Clock.schedule_interval(self.timer, refresh_time)

        with self.canvas:
            self.centered_circle = Line(circle = (self.center_x, self.center_y, 50), width = 2)

    def timer(self, dt):
        # Get data from serial port
        value = arduino.readline()

        # Draw the circle according to the data
        self.centered_circle.circle = (self.center_x, self.center_y,value)

        # More about drawing in Kivy here: http://kivy.org/docs/api-kivy.graphics.html
```

Esta clase Test1 sería la que se ejecutaría dentro del build de la estructura básica de un programa en kivy explicado anteriormente (Vease figura 5.1), tal y como se muestra en la siguiente imagen:

```
# Main App class
class SerialDataApp(App):
    def build(self):
        return Test1()

# Main program
if __name__ == '__main__':
    # Connect to serial port first
    try:
        arduino = serial.Serial('COM5', 115200)
        print "Connected"
    except:
        print "Failed to connect"
        exit()

    # Launch the app
    SerialDataApp().run()

    # Close serial communication
    arduino.close()
```

Tal y como se puede observar en la clase app principal de la aplicación “SerialDataApp”, lo único que hace es ejecutar la clase Test1 que contiene todo lo anteriormente expuesto.

En las últimas líneas de código, antes de lanzar la aplicación con “SerialDataApp().run()”, se ha de configurar y comprobar que hay algo conectado en el puerto USB principal del móvil/Tablet. Lógicamente hemos de saber el puerto que ya no será “COM5”, como en el caso del PC. Para obtener esta información se ha utilizado una aplicación llamada “Free USB Serial Term”. Ahí, en el apartado USB device descriptions, se puede identificar el nombre del puerto, que tal y como se puede apreciar en la imagen corresponde a: /dev/bus/usb/001/002.

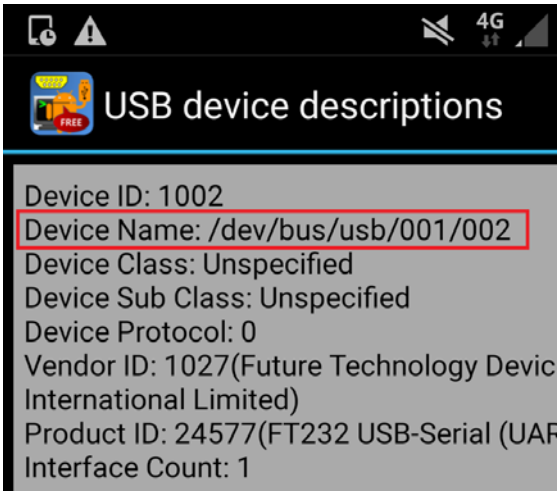

Figura 5. 2 - Descripción USB dispositivo. Fuente: Propia.

Por tanto, la línea de código de configuración del puerto USB queda de la siguiente forma:

```
try:
    arduino = serial.Serial('/dev/bus/usb/001/002', 115200)
    print "Connected"
```

5.1.2. Probando código USB Serial Kivy

Una vez escrito este código test para probar que realmente se puede leer datos es momento de testearlo en el móvil/Tablet. Si esta prueba resulta satisfactoria se procederá a adaptar el código Python anteriormente escrito a kivy y a integrar esto en el osciloscopio.

Para probar el código se utilizará una app llamada “Kivy Launcher”, que nos permitirá ejecutar el código sin necesidad de compilar, ni de dar permisos a la aplicación, etc… Es la forma más rápida de saber si realmente el código funciona. Una vez se tenga todo funcionando se puede plantear la opción de crear un archivo .apk (archivo de instalación de una aplicación Android). Para ello, se podría utilizar el “Buildozer”, que es una herramienta que automatiza todo el proceso de compilado del código. Como Buildozer funciona exclusivamente en Linux y en este caso se está utilizando Windows como sistema operativo, se instalará una máquina virtual mediante “Oracle” para simular un sistema Ubuntu de 64 bits que es el requerido por Buildozer y desde ahí se ejecutará la herramienta.

Para hacer uso de la aplicación Kivy Launcher se ha de crear una carpeta en la raíz del dispositivo en el que vamos a ejecutar el código, llamada “kivy”. Dentro se crea una carpeta con el nombre del proyecto (el mismo nombre que se le ha otorgado en el código) y se añade ahí el archivo main.py que contiene el código de la aplicación. Adicionalmente, se añade un archivo txt llamado “Android.txt”, en donde se escribirá información básica de la aplicación, tal como nombre (se recuerda que ha de ser el mismo que el otorgado en el código), autor del código, y la orientación de la aplicación (si se quiere que se vea en vertical, “portrait” o en horizontal, “landscape”).

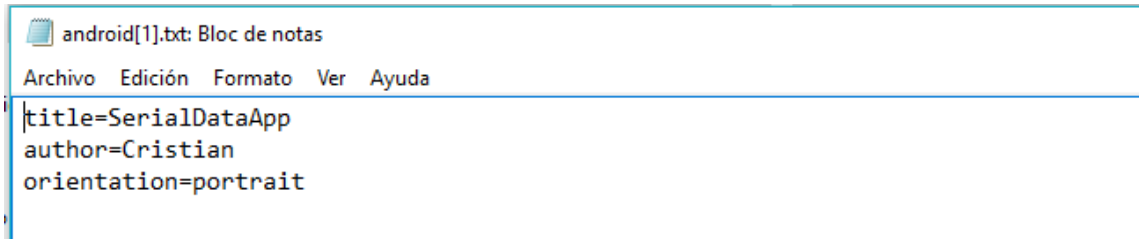

Figura 5. 3 - Archivo Android.txt. Fuente: Propia.

Se ejecuta la aplicación y el programa “SerialDataApp” y se obtiene el error que no se puede acceder al puerto USB aun habiéndolo configurado correctamente. Indagando se descubre que no se puede acceder desde un código escrito en Python al USB del terminal sin tener permisos especiales, es decir, si se quiere tener acceso a los datos recogidos por el puerto USB del móvil/tablet se ha de rootear. Lo que a simple vista no parece un problema insalvable se convierte en un impedimento suficiente para no cumplir los requisitos expuestos previamente del proyecto. El objetivo era crear un osciloscopio para un terminal para poder disponer de uno sin necesidad de instalar librerías, complementos, etc… es decir, algo que con solo ejecutar desde cualquier terminal funcionase. En este caso, si se quiere acceder a los datos del serial mediante USB el móvil/Tablet ha de estar rooteado, que no es algo genérico, por tanto, incumple un requisito básico del proyecto y no es viable continuar el proyecto por esta dirección. Se ha de intentar conseguir hacer una aplicación mediante un lenguaje de programación que no requiera rootear el dispositivo para acceder a los datos del serial mediante USB y eso lo ofrece por ejemplo Basic for Android (B4A) [véase el apartado 7].

6. Basic for Android (B4A)

6.1.1. Introducción

Basic for Android (B4A a partir de ahora) es un lenguaje de programación que incluye todas las características necesarias para desarrollar fácilmente cualquier tipo de aplicación para Android. B4A es una versión moderna del lenguaje de programación Visual Basic.

Una de las características claves a la hora de programar con B4A es que todas las aplicaciones compiladas por B4A son totalmente nativas, es decir, sin ninguna dependencia. Esto es realmente importante para solventar el problema que apareció con Python y su librería Kivy, que requería de permisos especiales para acceder al puerto USB del terminal. En este caso, este permiso no es necesario y, por tanto, se usará B4A para programar el osciloscopio.

6.1.2. Acceder modo Bit Bang desde terminal

El primer paso a ejecutar es conseguir un código que permita acceder al modo Bit Bang del Bus Pirate desde un terminal móvil. Para ello, se escribirá un código muy sencillo que al pulsar un botón, en este caso llamado “START”, permita entrar en ese modo. Lo primero que se define en un programa hecho a partir de B4A son los atributos de la aplicación, es decir, dar un nombre a la aplicación, definir qué orientaciones soporta, si se puede instalar en un dispositivo externo, etcétera.

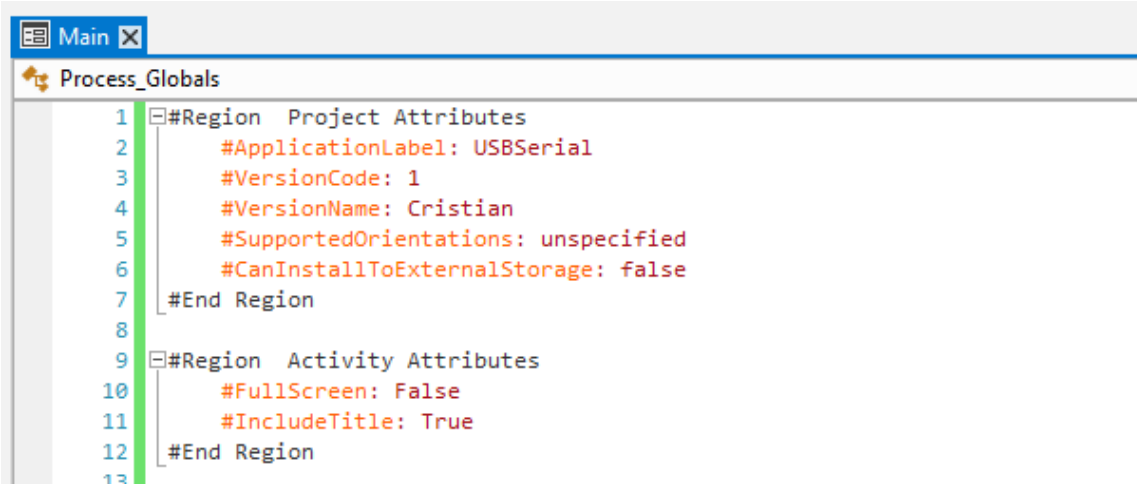

Figura 6. 1 - Atributos aplicación. Fuente: Propia.

Una vez definidos los atributos básicos se declaran las variables. En B4A se distinguen (y, por tanto, van declaradas aparte) entre variables globales de proceso (“Process global variables”) que se definen en el Sub Process_Globals y entre las variables de actividad que se definen en el Sub Globals. Las primeras son llamadas cuando el código empieza. Estas variables son públicas y, por tanto, pueden ser accedidas pro otros módulos también. Las variables de actividad son propiedad de la actividad en sí, cada vez que la actividad es creada se llaman estas variables y existen tanto como lo hace la actividad. Por ejemplo, para definir objetos del layout de la aplicación es necesario declarar estos objetos (etiquetas, botones, etc…) de este modo, ya que sólo han de aparecer cuando se inicie la actividad correspondiente y desaparecer tras finalizar la actividad.

Así pues el siguiente paso es diseñar el layout de la aplicación. En este primer test, lo único que se necesita es un botón START y un botón EXIT que salga de la aplicación. A través de la opción “designer” se diseña el layout:

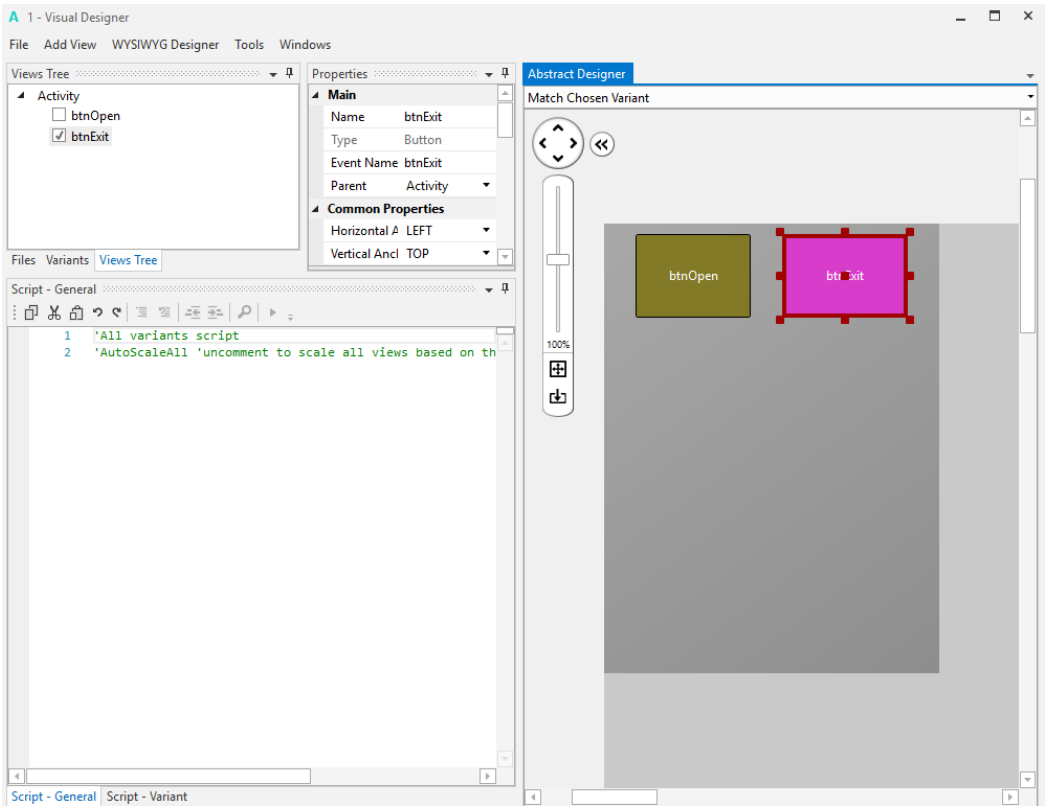

Figura 6. 2 - Visual designer con layout inicial. Fuente: Propia.

A través de la columna “Properties” se pueden cambiar diversas propiedades de los objetos tales como el color, tamaño, dependencia…

Se puede apreciar cómo tanto START como EXIT se han definido como botones, es decir, objetos que al pulsar sobre ellos realizan un proceso que se definirá más tarde en el código.

Una vez creado el layout mediante la opción “Generate Members” se genera todas las declaraciones de los objetos así como las esctructuras de las subrutinas correspondientes automáticamente.

El siguiente paso, es definir la subrutina “Activity_Create”, es decir, definir que se quiere que haga el programa nada más abrir la aplicación. Por ahora, lo único que ha de hacer es cargar el layout que se ha creado anteriormente. Así pues, la subrutina queda de la siguiente manera:

```
33  Sub Activity_Create(FirstTime As Boolean)
34      Activity.LoadLayout("1")
35  End Sub
```

Lo siguiente que se definirá será la acción que se ejecutará al pulsar el botón EXIT ya que es algo muy sencillo. Lo único que ha de hacer es salir de la aplicación. Por tanto, la subrutina del botón EXIT es:

```
149 Sub btnExit_Click
150     ExitApplication
151 End Sub
```

Ya sólo queda definir la subrutina del botón START. Aquí se ha de traducir el código antes escrito en Python. Para acceder al USB se ha de incluir en nuestro código la librería USBSerial (en este caso se ha utilizado la versión 2.40 de la misma) pudiendo acceder al USB mediante la clase UsbSerial y pudiendo interactuar con él a través de la clase AsyncStreams.

Para acceder al USB se declara una variable usb1 de la case UsbSerial. A través de usb1.UsbPresent se puede comprobar si hay un USB o no conectado al dispositivo. Una vez comprobado que hay algo conectado (si no lo hubiese aparecería un mensaje de error indicando que no lo está) mediante la opción usb1.RequestPermission se le pregunta al usuario si se le quiere dar permisos al puerto USB para leer a través de él. Si nos otorga este permiso, abrimos el USB a la velocidad que se quiera (en este caso, y tal y como se había hecho anteriormente, a 115200). Una vez abierto, ya se puede proceder a enviar bytes a la placa.

```
Sub btnOpen_Click
    If usb1.UsbPresent(1) = usb1.USB_NONE Then  ' Ver_2.4
        Log("Msgbox - no device")
        Msgbox("No USB device or accessory detected!", "Error")
        Log("Msgbox - returned")
        Return
    End If
    Log("Checking permission 1")
    If (usb1.HasPermission(1)) Then ' Ver_2.4

        Dim dev As Int
        dev = usb1.Open(115200, 1)      ' Ver_2.4
```

Definiendo astreams1 como una variable de clase AsyncStreams, si se inicializa esta variable (a través del comando astreams1.Initialize), se pueden enviar bytes a la placa mediante astreams1.write. Entonces, tal y como se especifica en las características del Bus Pirate, si se envía veinte veces el byte 0x00, la placa entrará en modo Bit Bang. Por lo tanto, se hace un loop en el que se envía la variable zbyte (que se ha declarado previamente como una variable de clase Byte, por lo que necesitaremos la libería ByteConverter, que no suele estar activada por defecto, activada) con el valor 0x00 veinte veces.

```
astreams1.Initialize(usb1.GetInputStream, usb1.GetOutputStream, "astreams1")

                    zbyte (0) = 0x00
                    For i=1 To 20
                        astreams1.Write(zbyte)
                    Next
```

Una vez escrito este código se puede observar como el Bus Pirate accede al modo Bit Bang una vez se pulsa el botón START de la aplicación.

6.1.3. Resetear Bus Pirate

Uno de los problemas que han surgido a la hora de trabajar con el Bus Pirate es el hecho de que una vez entra en modo Bit Bang aunque se cierre la aplicación y se vuelva a abrir la placa conserva el estado en modo Bit Bang. Se tenía que desconectar y volver a conectar el cable USB para poder resetear la placa a modo terminal de usuario para poder comprobar que entraba de nuevo en el modo Bit Bang. Para no tener que hacer esto cada vez que se quiera resetear la placa, se ha decidido implementar un botón de reset, que lo haga con tan solo pulsar el botón sin necesidad de tener que desconectar el cable.

Se añade un objeto botón más en el layout (“btnReset”) y se genera una subrutina para este botón. Si se observa en las especificaciones de la placa, si se le envía, estando en modo Bit Bang, el byte 00001111 (0x0F) la placa se resetea saliendo del modo Bit Bang y volviendo al modo terminal de usuario (se puede comprobar que realmente se ha vuelto al modo si el LED MODE de la placa se ha apagado nuevamente). Así pues la subrutina queda de la siguiente forma:

```
95  Sub btnReset_Click
96      zbyte (0) = 0x0D
97      astreams1.Write(zbyte)
98      Delay(1000)
99      zbyte (0) = 0x0F
100     astreams1.Write(zbyte)
101     astreams1.Close
102     usb1.Close
103
104 End Sub
```

6.1.4. Código USB Serial B4A

Actualmente se dispone de un aplicación con tres botones: el botón START, que hace que el Bus Pirate entre en modo Bit Bang, el botón RESET, que hace que una vez en modo Bit Bang se resetee la placa y, por tanto, vuelva al modo terminal de usurario, y el botón EXIT, que cierra la aplicación.

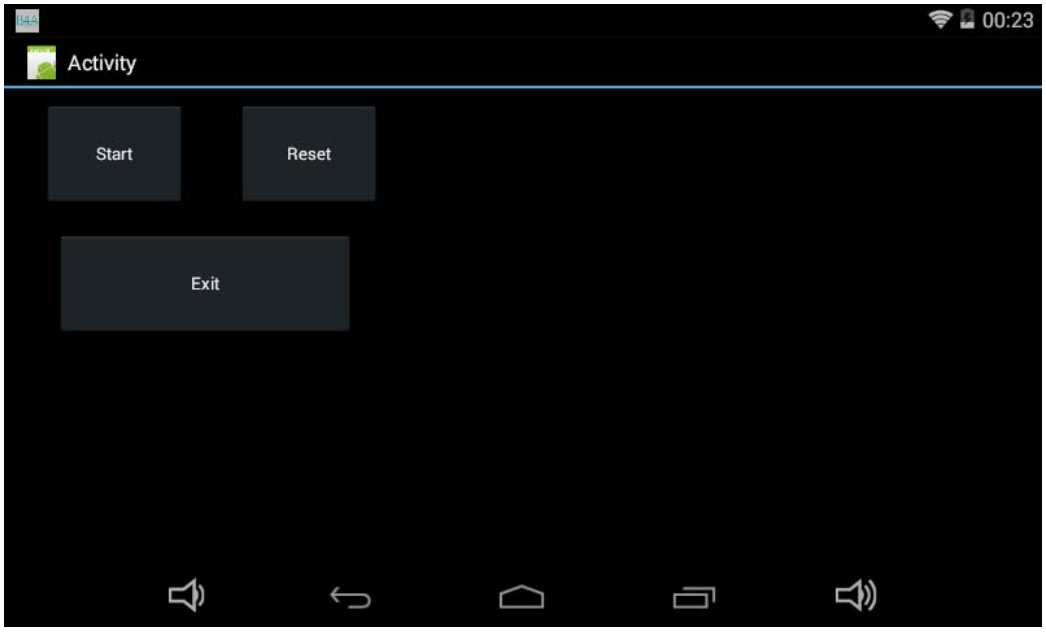

Figura 6. 3 - Layout aplicación USB Serial. Fuente: Propia.

El objetivo ahora es que cuando se pulse el botón START no sólo haga que la placa acceda al modo Bit Bang sino que la sonda ADC capture mediciones. Según las especificaciones de la placa, se puede hacer una medición si se envía el byte 0x14 a la placa o varias mediciones, tan rápidas como el UART lo permita, enviando el byte 0x15. Se empezará probando esta última opción. Así pues, después de enviar veinte veces el byte 0x00 para entrar en modo Bit Bang, y tras incluir un breve delay de 1s para que no se sature la placa, se enviará el byte 0x15:

```
80              Delay(1000)
81              zbyte(0)=0x15
82              astreams1.Write(zbyte)
83              flag=True
```

Se han de preparar los datos que llegan para poder ser utilizados luego en el osciloscopio. Para ello declaramos una variable booleana flag y la inicializamos como false. Se sabe que los datos que nos devuelve la placa al enviar tanto el byte 0x14 como el 0x15 es la medición de la sonda ADC en formato de 2 bytes, en donde los 8 bits más significativos van primero. Por tanto, esto se ha de tener en cuenta a la hora de tratar los datos. Se declara una variable data como un array de bytes de 2 en donde se irá guardando los datos obtenidos por el convertidor ADC. También se define otra variable USBValue como Float en donde se irá guardando el valor de data tratado para ser útil en el osciloscopio. Para el tratamiento de los nuevos datos que se van recibiendo del USB se debe escribir las líneas de código correspondiente en al subrutina Astreams1_NewData. Obviamente sólo se quiere que se ejecuten las instrucciones que se pondrán a continuación si se ha enviado el byte 0x15 para hacer mediciones y que, por tanto, se esté recibiendo datos. Entonces, lo primero que se hace es poner un condicional en que flag= true para ejecutar el resto (se recuerda que se declara flag como true una vez se ha enviado el byte 0x15 al Bus Pirate). Los datos vienen en forma de 2 bytes y para evitar que se mezcle, por ejemplo, el último byte del dato i-1 con el primer byte del dato i, se utilizará una variable Counter (declarada como Int e inicializada a 0) y se hará uso del buffer.

```
107 Sub Astreams1_NewData (Buffer() As Byte)
108     ' You must check for DeviceInfo or analyze Buffer data to know what is connected to the USB
109     ' The order of the USB could change as you plug them and could change when changing the hub port they are connected to
110
111     If flag=True Then
112
113         data(Counter)=Buffer(0)
114         If Buffer.Length > 1 Then
115             Counter= 1
116             data(Counter)=Buffer(1)
117
118         End If
119
120
121         Counter= Counter + 1
```

Una vez la variable Counter tenga el valor 2 significará que ya se tiene una pareja de bytes que nos ha enviado la placa. Por tanto, esos bytes se tratan siguiendo la expresión V= (ADC/1024)*6,6 para convertirla en Voltios y que sea de utilidad para el osciloscopio. Tras esto el contador Counter se pone a 0 para repetir el proceso por cada par de bytes enviados por el Bus Pirate.

```
118             If Counter == 2 Then
119                 USBValue= data(0)
120                 USBValue= USBValue*256
121                 USBValue= USBValue + data(1)
122                 USBValue= (USBValue/1023)*6.6
123                 Counter=0
124             End If
125             Log(USBValue)
126         End If
127 End Sub
```

Tras probar el código y ver el log se puede observar que al tomar la primera medición el programa da error, se queda bloqueado y no realiza más mediciones (también se puede ver que no está tomando mediciones porque el LED USB, que parpadea cada vez que se está leyendo una medición, no parpadea). Por tanto, se ha de buscar una forma alternativa para poder hacer mediciones sin que el programa se bloquee.

Para hacer esto, se enviará el byte 0x14 (que realizaba una medición de la sonda ADC) múltiples veces para simular que estamos enviando el 0x15. Para ello, se hará uso de una

variable Timer1 de clase Timer que lo que hace es ir repitiendo las instrucciones de dentro de su subrutina cada cierto tiempo, establecido al inicializar la variable.

Se ha hecho una prueba inicial con un Timer de 500 ms, es decir, la velocidad de muestreo es de 0,5 s para comprobar que se toman las mediciones. Una vez comprobado, se ha subido la velocidad hasta llegar a una velocidad de muestreo de 2 ms, que corresponde a 500 Hz, ideal para que se pueda utilizar el osciloscopio como aparato de medición de voltaje de red de 50 Hz (Hay que recordar que para capturar una señal con frecuencia f se recomienda muestrear a fx10).

```
Timer1.Initialize("Timer1", 2) ' 1000 = 1 segundo
```

A continuación se habilita la variable Timer1 dentro de la subrutina del botón START después de entrar en el modo Bit Bang.

```
Timer1.Enabled = True
```

Ahora, dentro de la subrutina Timer1_Tick, se escribe el proceso que se desea repetir mientras la variable Timer1 esté habilitada o se cierre la aplicación. En este caso, enviamos a la placa el byte 0x14 tal y como se ha hecho anteriormente.

```
Sub Timer1_Tick

    zbyte(0)=0x14
    astreams1.Write(zbyte)
End Sub
```

Ahora se tiene todo preparado para que al pulsar el botón START de la aplicación creada, la placa entre en modo Bit Bang y vaya midiendo a través de la sonda ADC voltajes (que serán dados en forma de pares de bytes pero que son debidamente transformados a Voltios). Se han introducido un log del valor USBValue para que se muestree en el registro estas mediciones y comprobar que efectivamente se están tomando medidas y se están conversando adecuadamente. Para ello haremos medidas a tierra (0 V) y al pin de 5V para comprobar que es correcto.

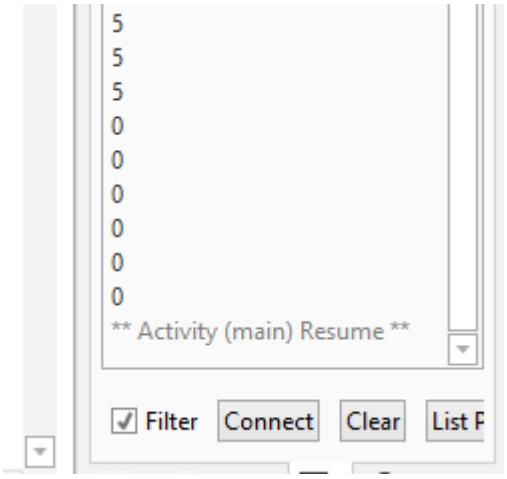

Figura 6. 4 - Valores de tensión. Fuente: Propia

Efectivamente, si nos fijamos en los valores que muestra el registro, se puede corroborar como el código escrito funciona. Ahora sólo queda integrar este valor en el osciloscopio.

7. Osciloscopio

Ya se dispone de un código que nos permite conectar el Bus Pirate mediante USB a un dispositivo móvil o Tablet Android y que a través de una aplicación ponga la placa en modo Bit Bang y realice mediciones a través del convertidor ADC tratando los datos recibidos para convertirlos en Voltios. Lo último que queda es integrar este valor en el osciloscopio que se ha tomado como base para completar así el objetivo del proyecto.

El osciloscopio que se ha escogido como base para el del proyecto tiene varias características que han sido descartadas por ser innecesarias o por no ser de interés. En el osciloscopio original el programa generaba cuatro curvas, los valores de las cuales eran funciones seno que estaban desfasadas entre ellas. Para el proyecto se necesita una sola curva, ya que sólo se mide una variable, por lo que las otras tres se pueden suprimir. Por supuesto, el valor de la curva, en vez de ser una función seno, tomará los valores que le vaya proporcionando la placa mediante el código antes escrito. Por tanto, lo primero que se hace es ajustar las escalas a los valores que puede tomar la curva.

El modo en el que se trabajará será en el modo “Single Shot”, en el que se toman valores de la curva y se dibuja una vez se han tomado todos los valores. En este caso, se tomarán 100 valores y una vez obtenidos se dibujará la curva con éstos. Por tanto, se integra el código USB Serial en el osciloscopio (incluidas las subrutinas necesarias) y se procede a modificar la subrutina Astreams1_NewData de tal forma que se comporte tal y como se desea.

```
USBValue= USBValue + data(1)
If SingleShot== True Then
    'Log(USBValue)
    CurveVal(0, ii) = (USBValue/1023)*6.6
    'Log(CurveVal(0,ii))
    't= t+dt
    'xx= xx+ dx
    xxx (ii+1)= xxx(ii)+ dx
    ii= ii+1
    'DrawCurves
    If ii > 100 Then
    DrawCurves2
    End If
```

Después de haber obtenido el valor del voltaje medido por la sonda ADC de la placa, se pone como condición el estar en modo SingleShot. Si es así se le van otorgando a la variable CurveVal en donde el argumento 0 es la curva (que ya hemos dicho que es única) y ii son los distintos valores por los que se irá iterando. Se le van dando valores a CurveVal (que corresponde al eje Y) mientras se va actualizando también el valor del eje X (representado por la variable xxx) hasta llegar a las 100 iteraciones en donde se llama a la función DrawCurves2 en que se dibuja la curva.

Se hace la prueba y se miden diferentes pines de la placa para comprobar que efectivamente al tomar 100 valores de voltaje, la aplicación te dibuja la curva:

Figura 7. 1 - Resultado muestre osciloscopio. Fuente: Propia.

8. Presupuesto

En este apartado se realizará una valoración económica del proyecto. Obviamente el mayor impacto a nivel de coste del proyecto recaerá en la contratación de un ingeniero que estudie, analice y desarrolle el proyecto. Aun así, si hay algunos costes de material, como la compra de la placa Bus Pirate o la compra de la licencia del programa en el que se realice el código.

Así pues si se realiza la suposición de que el ingeniero dispone de los conocimientos de programación necesarios para el desarrollo del código y no se necesite otro que lo haga, se podrían resumir sus tareas en las siguientes:

- Estudio del problema. Fase en que el ingeniero estudia el problema a resolver. Esto requiere de un estudio de mercado así como un estudio del arte, es decir, el estudio del estado de esta temática actualmente y sus avances.
- Estudio de la placa. Fase caracterizada por el estudio de las especificaciones de la placa que se usará en el proyecto. Es un conocimiento que raramente tendrá el ingeniero ya que cada placa tiene unas especificaciones únicas y es probable que nunca se haya trabajado con ella.
- Diseño del algoritmo. Fase en la que el ingeniero diseña el/los algoritmos necesarios para realizar el proyecto y determinar la estrategia a seguir a la hora de programarlos para satisfacer los requisitos.
- Programación del algoritmo. Básicamente se basa en la escritura del código diseñado en la anterior fase.
- Testeo del código y depuración. Esta fase consiste en hacer varias pruebas en diferentes condiciones para comprobar que el código funciona correctamente y cumple los requisitos propuestos.
- Análisis de resultados. El ingeniero realiza un análisis de resultados tras hacer el proyecto para determinar que efectivamente se cumple con los objetivos del mismo.

Si se asocia un tiempo medio a cada una de estas fases y tomando como base salarial la de un ingeniero con un mínimo de un año de experiencia se puede determinar el coste de mano de obra del proyecto.

Grupo	Escala Salarial de Ingreso 2016				
	Grado	Ingreso	1.er año	2.º año	3.er año
Ingeniero/a	8	23.395	26.514	29.112	34.567

Figura 8. 1 - Salario Ingeniero BOE. Fuente: BOE 2016

Para saber el salario/hora sería conveniente dividir el salario anual entre las 1722 horas/año de jornada anual predeterminada. Ese salario/hora será el que se multiplicará por el total de horas trabajadas, calculando así el coste: 26.514/1722= 15,4 €/h

Más allá del coste de mano de obra del proyecto existe un pequeño coste adicional de la compra de la placa Bus Pirate así como de la licencia del programa en el que se desarrolla el código que, en este caso, es B4A.

Así la tabla de precios quedaría de la siguiente manera (los precios incluyen el IVA):

	Tarea	**Tiempo (h)**	**Sueldo (€/h)**	**Coste (€)**
Ingeniero con mínimo un año de experiencia	Estudio del problema	20	15,4	308
	Estudio de las especificaciones de la placa	5	15,4	77
	Diseño del algoritmo	60	15,4	924
	Programación del algoritmo	40	15,4	616
	Testeo código y depuración	30	15,4	462
	Análisis de resultados	10	15,4	154
TOTAL				**2541**

Tabla 8. 1 - Costes mano obra. Fuente: Propia

Producto	**Descripción**	**Cantidad**	**Coste (€)**
Placa	Placa Bus Pirate v3.6a	1	36,86
Cable USB	Cable USB mini-B	1	3,87
Cable Bus Pirate	Conector principal Bus Pirate 2x5 pines	1	5,57
B4A	Licencia estándard B4A	1	59
TOTAL			**105,3**

Tabla 8.2 – Costes material proyecto. Fuente: Propia

Por tanto, el coste total del proyecto sería de **2646,3 €.**

9. Impacto ambiental

Actualmente, y dada la situación mundial de cambio climático, es muy importante realizar un estudio de impacto ambiental de cualquier proyecto que se realice para poder ver qué consecuencias tendría el desarrollo del mismo sobre el medio ambiente.

En un primer momento, se podría pensar, que los tipos de proyecto que son de este tipo, proyectos basados casi exclusivamente en la programación de código, podrían ser indiferentes respecto al impacto que tienen sobre el medio ambiente. Y aunque realmente no se esté fabricando nada, sí que pueden suponer un impacto indirecto.

Si se piensa en los aspectos negativos que podría tener un proyecto como este en el medio ambiente podría ser el uso de una placa electrónica que sí que tiene un impacto por el tipo de material de la que está hecha, por ejemplo, cobre, cromo, etcétera. El uso de estas placas, al ser desechadas, supone una contaminación de áreas verdes, de ríos, lagos y mares, de emisiones a la atmósfera de elementos tóxicos y de generar desequilibrio de los ecosistemas.

Por otra parte, como aspectos positivos, se puede destacar que más allá del uso de una placa electrónica (que, además, es de un tamaño bastante reducido por lo que contiene menos elementos contaminantes) no genera más contaminantes. Pensando desde un punto de vista indirecto, si se lograse realizar un osciloscopio totalmente funcional y que pudiese sustituir al físico, sería reemplazar una gran cantidad de elementos electrónicos contaminantes que contiene un osciloscopio físico por sólo una pequeña placa electrónica lo que obviamente sí tendría un impacto beneficioso para el medio ambiente. Obviamente se está hablando de una situación casi utópica, ya que la versión que se ha hecho del osciloscopio es muy básica y realmente no todas las funciones del osciloscopio podrían ser procesadas por una placa tan básica como Bus Pirate. Aunque en mediciones que no requieran de gran complejidad sí se podría acabar sustituyendo por la aplicación y aportando, por tanto, un impacto positivo.

Conclusiones

Tras el desarrollo del proyecto se pueden extraer las siguientes conclusiones.

Del estudio de la placa Bus Pirate se ha podido comprobar como una placa en apariencia sencilla puede dar mucho de sí. Lo primero que se puede concluir del estudio de la placa es que de los dos modos de los que dispone (modo terminal de usuario y modo de acceso binario), el primero de ellos, el modo terminal de usuario convendría ser usado cuando la velocidad de transmisión de datos no sea un factor determinante en el proyecto para el cual se esté utilizando ya que en este modo los datos se transmiten en texto y hace muy lenta su lectura. Es un modo ideal para probar las funcionalidades de la placa. Por otro lado, el modo de acceso binario es un modo en el cuál el envío se realiza mediante números binarios haciendo la transmisión de datos mucho más rápida y, por tanto, muy útil cuando en proyectos en los cuales la velocidad de transmisión de datos entre placa y terminal sea un factor clave, como es en el caso de un osciloscopio.

A la hora de escribir el código para realizar la comunicación entre USB y serial de la placa se han encontrado problemas de permisos para acceder al USB del terminal. Este es un factor muy importante y que se debe tener en cuenta si lo que se busca es accesibilidad. Python es un lenguaje de programación en el que no nos daba permiso a acceder al USB del móvil. B4A, sin embargo, sí nos permitía ese acceso. Por tanto, si se desea realizar un proyecto en el que sea necesario el uso del puerto USB de un dispositivo Android el lenguaje más eficiente para realizar una aplicación para dicho dispositivo sería B4A o directamente Java.

El objetivo de escribir un código que permita el envío de datos, previamente convertidos para ser de utilidad al osciloscopio, se ha completado con éxito. Además, este muestreo se ha podido hacer a una frecuencia de 500 Hz (2ms) que permitirá, al osciloscopio, medir señales de voltaje de red (50 Hz) que era una posible utilidad del mismo. El código de la comunicación entre USB y serial de la placa se ha integrado con éxito en una versión muy básica de un osciloscopio. Este osciloscopio opera en un modo llamado “Single Shot”, en el que muestrea una señal a una velocidad determinada (2ms en este caso) un número finito de veces (100 en este caso) y una vez recogidos todos los datos dibuja una curva utilizándolos.

Obviamente, en esta última parte del proyecto es donde habría una oportunidad de mejoría enorme del mismo. Hacer un osciloscopio con más funciones que la de “Single Shot” podría ser una de las posibles mejoras. Modos como hacer esta medición a tiempo real, es decir, dibujar en paralelo la curva mientras se están tomando los datos sin necesidad de esperar que se haga un muestreo finito.

También una posible mejora o ampliación del proyecto sería utilizar diferentes protocolos del utilizado, por ejemplo, I2C, SPI, etcétera.

Concluyendo, los objetivos del proyecto se han cumplido satisfactoriamente centrándonos sobre todo en el desarrollo del código de comunicación entre USB y serial de la placa. Aun así, el proyecto es ampliamente mejorable, sobre todo, en la parte de implementación en el osciloscopio, en el que el desarrollo de un osciloscopio más sofisticado y complejo podría ser la base para un posible futuro proyecto.

Bibliografía

Referencias bibliográficas

[1] Seagrave Wyken, *B4A: Rapid Android App Development Using BASIC*. 2015

[2] Klaus Christl, Erel Uziel, *B4A: Beginner's guide.* 2015

[3] Klaus Christl, Erel Uziel, *B4A: User's guide.* 2016

Bibliografia complementaria

- https://learn.sparkfun.com/tutorials/bus-pirate-v36a-hookup-guide
- http://dangerousprototypes.com/docs/Bus_Pirate_I/O_Pin_Descriptions
- http://dangerousprototypes.com/docs/Bus_Pirate
- http://dangerousprototypes.com/docs/Bus_Pirate_self-test_guide
- https://kivy.org/docs/gettingstarted/intro.html
- https://docs.python.org/3/tutorial/
- http://kio4.com/b4a/index.htm#indice